UM EECS 270 F22
Introduction to Logic Design

7. Binary Arithmetic

addition of unsigned number

Binary Arithmetic

- Representation of positive numbers (old news)
- Binary addition
- Representation of negative numbers
- Binary subtraction
- (Basic) Binary multiplication

Binary Addition

- How do we add two binary numbers?

Just like elementary school!

Decimal:

```
  1   1
  4 2 5 9
+ 1 8 3 7
-------
  6 0 9 6
```

This is the *carry out* of the first column, which becomes the *carry in* to the second column

the only different between adding decimal number and binary number is we only have 2 digits instead of 10.

Binary:

```
    1 1
  1 0 0 1
+ 0 0 1 1
---------
  1 1 0 0
```

- If we have a fixed number of bits (which is usually the case), a carry out of the most significant column indicates that there's not enough bits to hold the sum value. This case is referred to as **overflow.**

Addition Implementation

C
A
B
S

模拟每一位的计算.

- Addition of two 1-bit binary numbers, A and B – requires two output bits, which we'll call S (sum) and C (carry).

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

two bit answer

$$C = AB$$

只有两个都是1的时候才会进位.

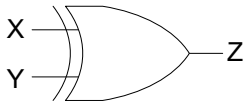
$$S = A\overline{B} + \overline{A}B = A \oplus B$$

XOR

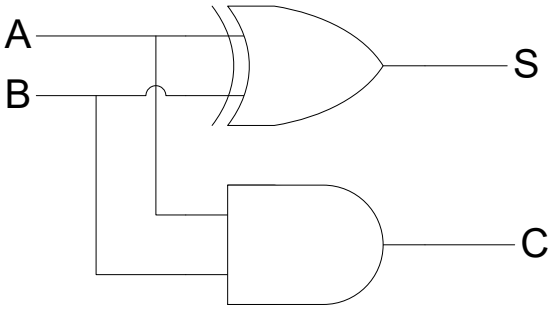
当A和B相同时不管同0还是同1 Sum都是0. 所以Sum只有在AB不同时才为'1'

XOR Operation

$$X \oplus Y \equiv X\overline{Y} + \overline{X}Y$$

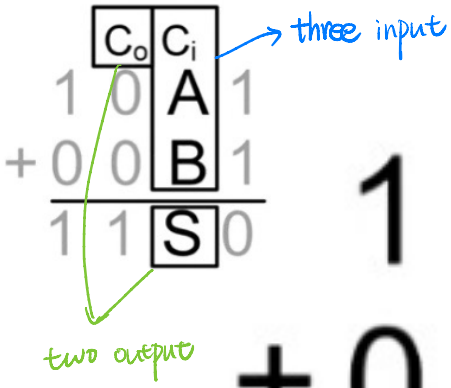


Implementation:



This circuit is called a **Half Adder (HA)**

- What if we want to add larger numbers?
  - What's missing from HA? 将HA的结论向外推导



$$\begin{array}{r} {\scriptstyle 1\ 1} \\ 1\,0\,0\,1 \\ +\,0\,0\,1\,1 \\ \hline 1\,1\,0\,0 \end{array}$$

相加HA，我们少了一个 Carry in.

Sum: HA是 $A \oplus B$ 表达

FA是 $A \oplus B \oplus C$

Carry o: HA是AB，现在新引入了一个 input C_i

$AB + AC_i + BC_i$ 进一步化简

$(A+B = A \oplus B + AB)$

$= A'B + AB' + AB$

$= B(A'+A) + AB'$

$= B + AB'$

$= (B+A)(B+B')$

$= B + A.$

$AB + (A \oplus B)C_i$

A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Need to add carry-in (C_i) input!

$$S = \sum_{A,B,C_i}(1,2,4,7)$$

$$C_o = \sum_{A,B,C_i}(3,5,6,7)$$

- Sum output:

$$S = \sum_{A,B,C_i}(1,2,4,7)$$

$$S = A'B'C_i + A'BC_i' + AB'C_i' + ABC_i$$

xNor xOr

$$S = C_i\left(A'B' + AB\right) + C_i'\left(A'B + AB'\right)$$

$$S = C_i\left(A \oplus B\right)' + C_i'\left(A \oplus B\right)$$

$$S = A \oplus B \oplus C_i$$

- Carry-out output:

$$C_o = \sum_{A,B,C_i}(3,5,6,7)$$

$$C_o = A'BC_i + AB'C_i + ABC_i' + ABC_i$$

$$C_o = A'BC_i + AB'C_i + ABC_i' + ABC_i + ABC_i + ABC_i$$

BC_i AC_i AB

$$C_o = AB + AC_i + BC_i$$

This requires 3 2-input AND gates and 1 3-input OR gate.

Can we do any better?

- Carry-out output: A bit of cleverness …

证: $A+B = (A \oplus B) + AB$

$AB + A\bar{C}_i + B\bar{C}_i$

$= AB + \bar{C}_i(A+B)$

$= AB + \bar{C}_i(A \oplus B) + AB$

$= AB + \bar{C}_i(A \oplus B) + ABC_i$

$= C_i(A \oplus B) + AB(1 + C_i)$

$= AB + (A \oplus B)C_i$

First, we observe that:

$$A + B = (A \oplus B) + AB$$

$$\begin{aligned} C_o &= AB + AC_i + BC_i \\ &= AB + C_i(A + B) \\ &= AB + C_i\big((A \oplus B) + AB\big) \\ &= AB + (A \oplus B)C_i + ABC_i \\ &= AB + (A \oplus B)C_i \end{aligned}$$

$AB + ABC_i = AB(1 + C_i) = AB$

if $AB = 0$ then $ABC_i = 0$

and, or, xor, and

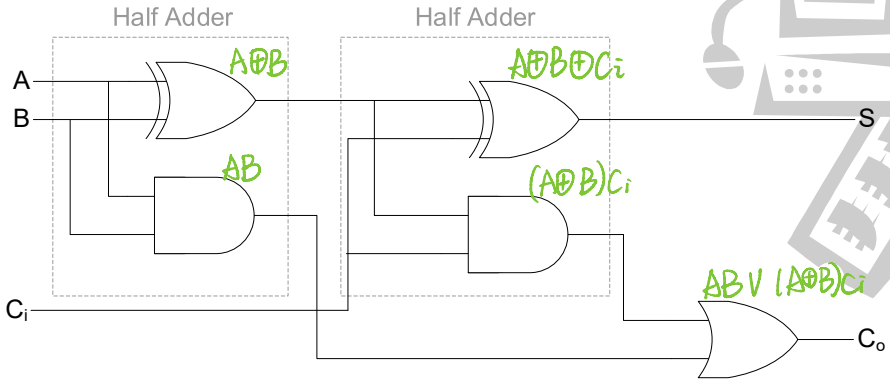
new gate we need.

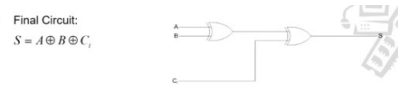
This requires 2 2-input AND gates, a 2-input OR gate, and a 2-input XOR, *but we've already implemented* $A \oplus B$ *for the Sum output!* So we only require 3 additional gates..

- Final Circuit:

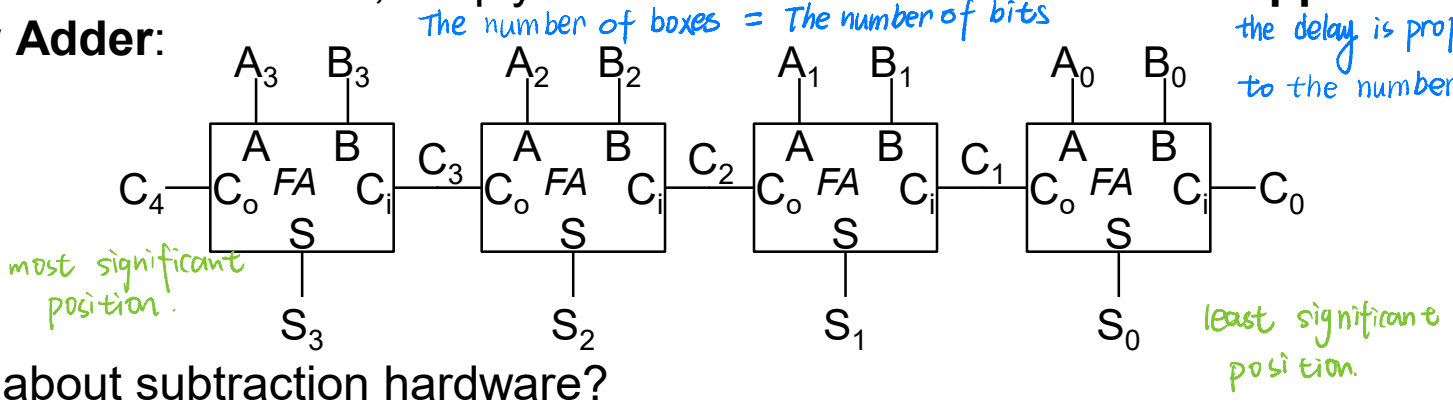
$$S = A \oplus B \oplus C_i$$

$$C_o = AB + (A \oplus B)C_i$$





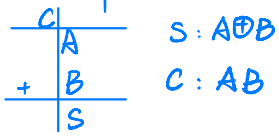
- This circuit is called a **Full Adder (FA)**
- After all that design work, we really just have 2 HAs with an OR gate
- To make an n-bit adder, simply cascade Full Adders to make a **Ripple Carry Adder**:



- What about subtraction hardware?
  - Could design subtractor hardware using process similar to adder design
  - Simpler way: re-use our addition hardware! A – B = A + (– B)

(– B)*???* *How do we represent* ***negative numbers?***

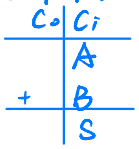
① Half Adder



S : A⊕B

C : AB

② Full Adder



S : A⊕B⊕Ci

C : AB+(A⊕B)Ci

由两个 Half Adder 和一个 or gate 组成

③ Ripple carry adder

n bits 的数相加减

需n个 Full Adder

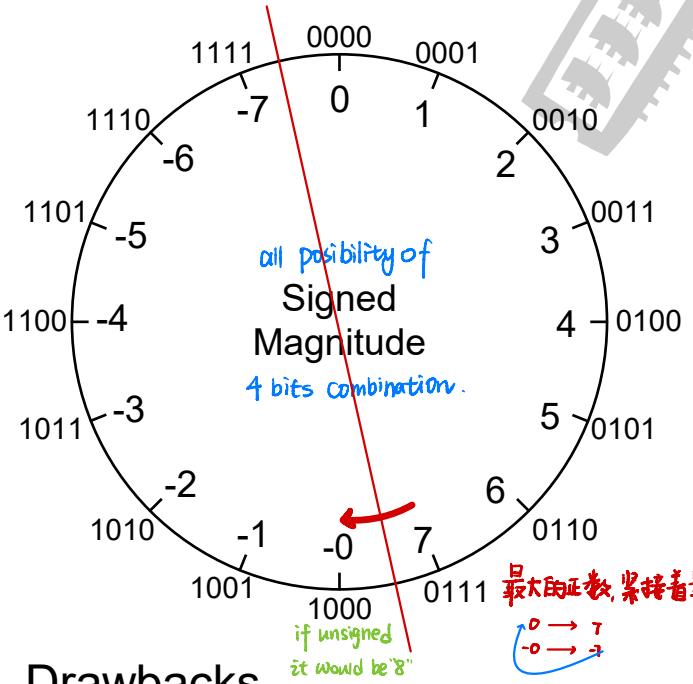
Binary Representation of Negative Numbers

- **Signed Magnitude:** Numbers consist of a magnitude and a symbol indicating whether the number is positive or negative
  - We're used to this: (-10: "-": sign, "10": magnitude)
- In binary, reserve MSB to represent the sign: 0 indicates a positive number, 1 indicates a negative number
- Sign bit position has no weight
- What is the range of numbers that can be represented with 4-bit binary signed-magnitude representation

  [-7 : 7]

- What is the range of numbers that can be represented with *n*-bit binary signed-magnitude representation?

  $[-(2^{n-1}-1) : 2^{n-1}-1]$



- Drawbacks
  - Two representations of 0
  - Signed magnitude arithmetic requires comparing sign bits and then performing addition or subtraction – leads to complex hardware

Can we do it without checking the sign?

Complement Representations

- **1s' Complement Representation:** MSB has weight of $-(2^{n-1}-1)$
- To get the representation of a negative number, write the positive representation and complement all bits

  Example: -3: 3 = 0011, -3 = 1100 = 1*(-7) + 1*4 + 0*2 + 0*1

- MSB is 0 if number is positive, 1 if number is negative
- What is the range of numbers that can be represented with *n*-bit 1s' complement representation?

$$[-(2^{n-1}-1) : 2^{n-1}-1]$$

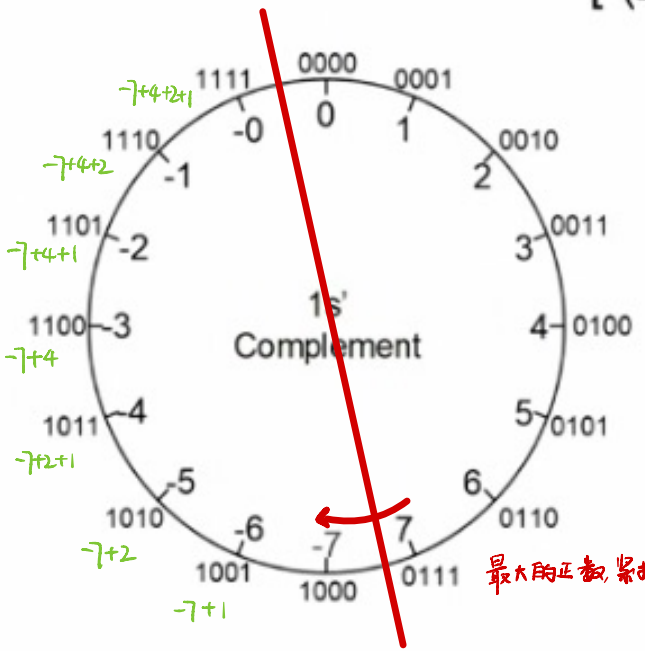
范围取决于最大值和最小值.

最大值: 0 1 1 1 ··· 1 ($n-1$ 个) $= 2^{n-1}-1$

最大值: 1 0 0 0 ··· 0 $= -2^{n-1}+1$

↑ 这个 MSB 的 weight 不再是 2^{n-1} 了, 而是 $-(2^{n-1}-1)$ 了.

和 Sign magnitude 的范围完全相同



最大的正数, 紧接最小的负数

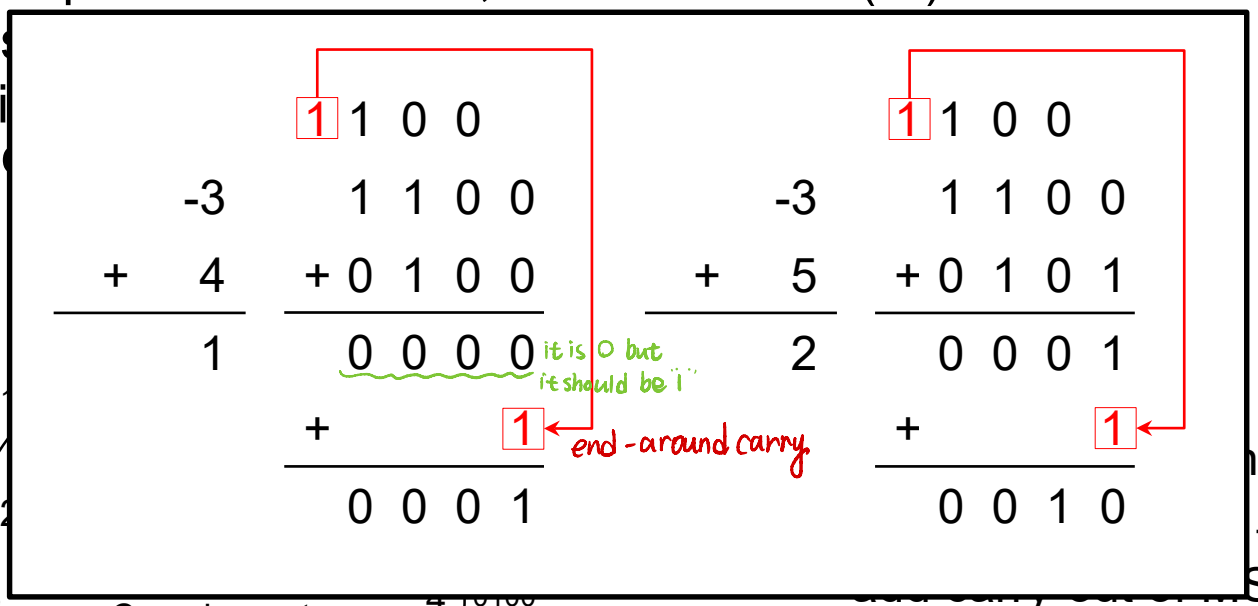
- Drawbacks
  - Two representations of 0
  - "Messy" arithmetic – must add carry-out of MSB to sum (**end-around carry**)

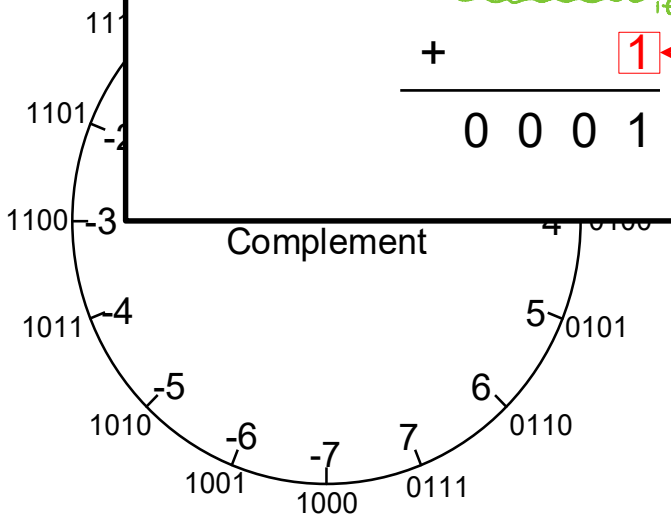
UM EECS 270 Fall 2022 11

Complement Representations

- **1s' Complement Representation:** MSB has weight of $-(2^{n-1}-1)$
- To get the representation of a negative number, write the positive representation and complement all bits

  Example: -3: 3 = 0011, -3 = 1100 = 1*(-7) + 1*4 + 0*2 + 0*1
- MSB is
- What i *n*-bit 1s' compl





ns of 0

– must

SB to sum (**end-around carry)**

Abandoned in the 60s in favor of 2's complement….

1's MSB weight is $(-2^{n-1}-1)$

- **2's Complement Representation:** MSB has weight of -2^{n-1}
- To get the negative representation of a number, write the positive representation, complement all bits, and add 1
  (ignoring any carry-out of the most significant column)

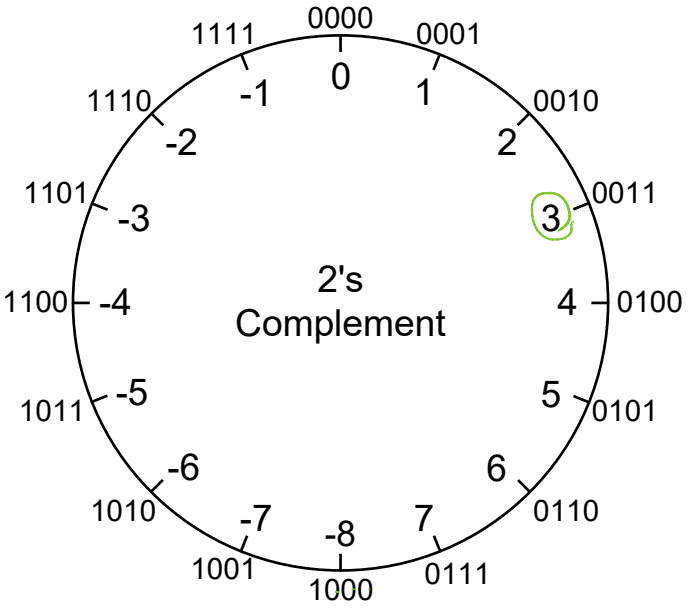
  Example: -4: 4 = 0100
  1011+1 = 1100
  = -8*1 +4*1 + 0*2 + 0*1 = -4

  the trick over here is we get around the end-around carry by doing the addition when we're doing the implement
- MSB is 0 if number is positive, 1 if number is negative
- What is the range of numbers that can be represented with *n*-bit 2's complement representation?

  $[-(2^{n-1}) : 2^{n-1}-1]$

我们解决了有两个零的问题，但新的状态是 range is not symmetry (one more negative)

但因为如果有 n bits，就有2^n种组合，且 negative number 在零左侧，positive number 在零右侧，则若只有1个零，则正负数数量一定不相同，所以负数比正数多1个



Complement of -8 (1000):

0111 + 1 = 1000

这是一个笔误，我们想要的是"8"，但这里并没有-8的expression方法。但这并不是一个严重的问题，因为-8在 very end of the number range，并不时常发生。

Complement of 0 (0000):

1111 + 1 = 0000 ✓

2's Complement Arithmetic

使用要求

- To add two numbers (positive or negative), use normal binary addition and *ignore carry out of most significant column*
- Addends and sum should always have the same number of bits

3: 0 0 1 1

-3 : (0011)' + 1 = 1100 + 1 = 1101 = -8+4+1 = -3

(-3) + 4

```
  1 1
  1 1 0 1
+ 0 1 0 0
  0 0 0 1
```

5 + (-7)

```
    1
  0 1 0 1
+ 1 0 0 1
  1 1 1 0
```

(-2) + (-3)

```
1 1
  1 1 1 0
+ 1 1 0 1
  1 0 1 1
```

- To subtract, take complement of subtrahend and add to minuend:

$$A - B = A + (-B)$$

  - Step 1: Generate (-B) – how?
  - Step 2: Add - use ripple-carry adder!

Subtraction Using RCA

- Step 1: Generate (-B)
  - Complement bits of B
  - Add 1
- Step 2: Add A to (-B) with RCA - *use carry-in of LSB!*

我们通过最开始的 Carry in 来实现 implement 在加法的操作.

Subtract 2 from 6 (6-2):

```
+2:  0 0 1 0
                 complement
     1 1 1  1 ← carry in
   0 1 1 0
 + 1 1 0 1
 -------------
   0 1 0 0
```

Subtract (-3) from 4 (4 – (-3)):

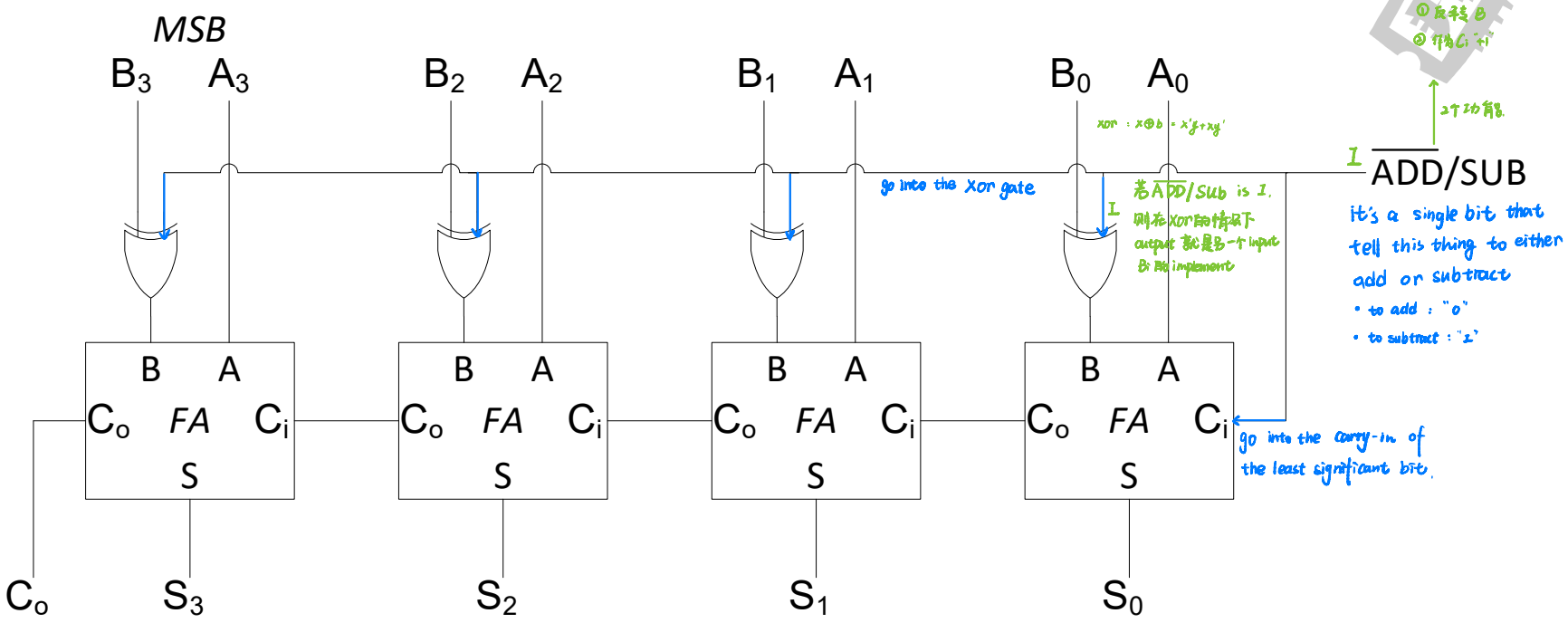
```
-3:  1 1 0 1
                 complement
         1
   0 1 0 0
 + 0 0 1 0
 -------------
   0 1 1 1
```

对之前的 ripple-carry adder 的改进,以适应减法运算.

- What modifications must be made to a ripple-carry adder to make a ripple-carry adder/subtractor?
  - Need a signal (C_2) to determine whether we're adding or subtracting!
  - Complement bits of B (conditionally) → XOR
  - Add 1 to B (conditionally) → use carry-in to LSB

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

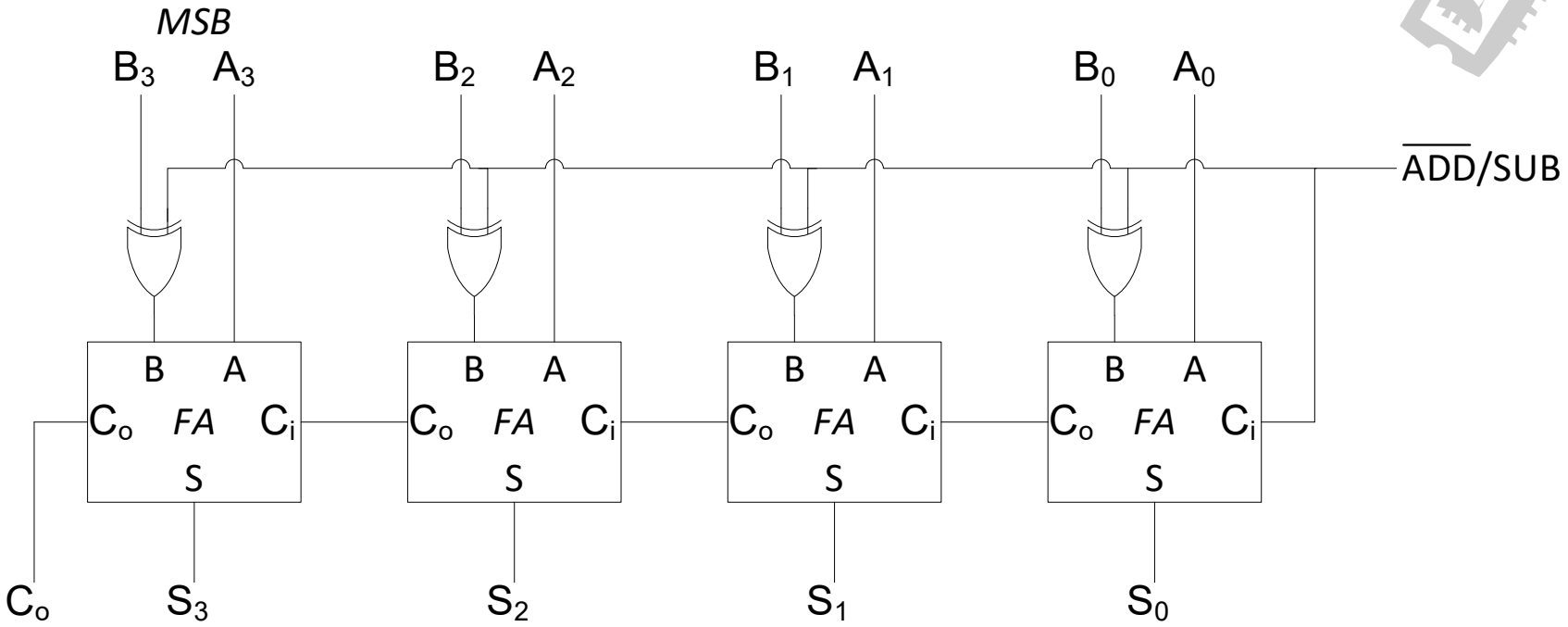
Ripple-carry Adder/Subtractor



$\overline{ADD}/SUB = 0 \rightarrow S = A + B$

$\overline{ADD}/SUB = 1 \rightarrow S = A - B$

Ripple-carry Adder/Subtractor



$\overline{ADD}/SUB = 0 \rightarrow S = A + B$

$\overline{ADD}/SUB = 1 \rightarrow S = A - B$

Overflow

- If result of operation is too large for the current number of bits, must notify user that result is invalid

5 + 6

```
 0 1   different
  0 1 0 1
+ 0 1 1 0
  1 0 1 1  = -5
  ↑
sign bit
```

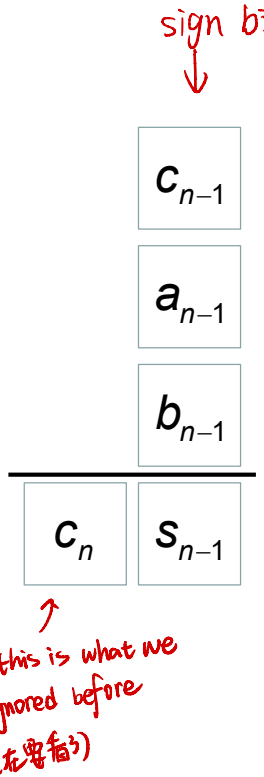
(-3) + (-6)

```
 1 0   different
  1 1 0 1
+ 1 0 1 0
  0 1 1 1  = 7
```

- Addition of two numbers with different signs can *never* overflow
- If sign bits of addends are the same and *different* from the sign bit of the result, then overflow has occurred
- Equivalently, if the carry-in to the most-significant column is different from the carry-out of the most-significant column, overflow has occurred
  - Easily to implement with an XOR gate

Overflow Detection

ovf = sign bits of addends are the same and *different* from the sign bit of sum



$$s_{n-1} = a_{n-1} \oplus b_{n-1} \oplus c_{n-1}$$

$$c_n = a_{n-1}b_{n-1} + c_{n-1}(a_{n-1} \oplus b_{n-1})$$

$$b_{n-1} = a_{n-1}$$

相加的两数 sign 相同

$y + oy' = y$

$xy + xy'$

$$s_{n-1} = a_{n-1} \oplus a_{n-1} \oplus c_{n-1} = 0 \oplus c_{n-1} = c_{n-1}$$

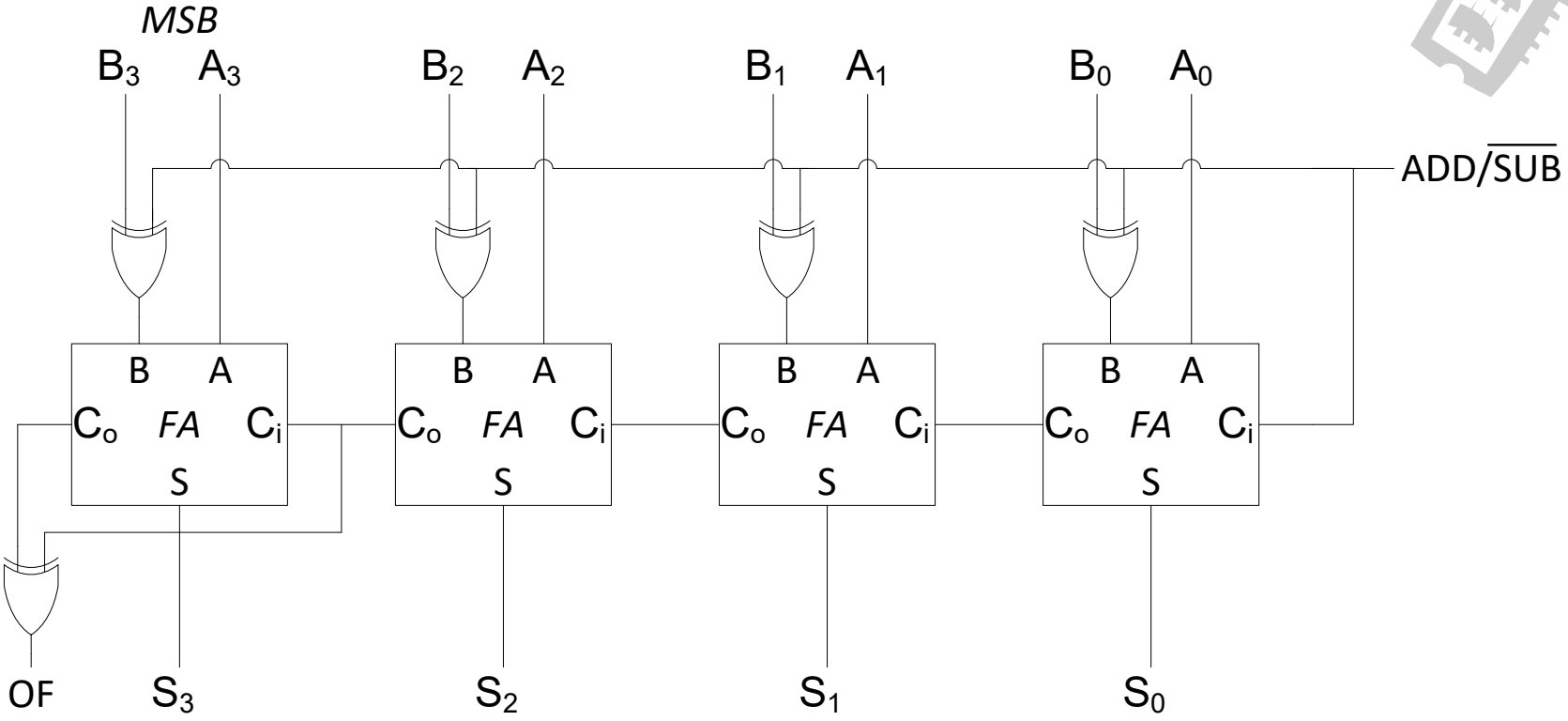
$$c_n = a_{n-1}a_{n-1} + c_{n-1}(a_{n-1} \oplus a_{n-1}) = a_{n-1} + c_{n-1} \cdot 0 = a_{n-1}$$

$$ovf = a_{n-1} \oplus s_{n-1}$$

$$= c_n \oplus c_{n-1}$$

the carry out — the carry in the MSB

Ripple-carry Adder/Subtractor with Overflow Detection



$\overline{ADD}/SUB = 0 \rightarrow S = A + B$

$\overline{ADD}/SUB = 1 \rightarrow S = A - B$

OF = 1 → Overflow

Ones' v. Two's Complement

$$X_{1C} = x_3 x_2 x_1 x_0$$

$$-X_{1C} = x_3' x_2' x_1' x_0'$$

x	$1 - x = x'$
0	$1 - 0 = 1$
1	$1 - 1 = 0$

$$\begin{array}{rcccc} & 1 & 1 & 1 & 1 \\ - & x_3 & x_2 & x_1 & x_0 \\ \hline -X_{1C} = & x_3' & x_2' & x_1' & x_0' \end{array}$$

$$-X_{1C} = 15 - X_{1C}$$

$$-X_{1C} = \left(2^n - 1\right) - X_{1C}$$

$$X_{2C} = x_3 x_2 x_1 x_0$$

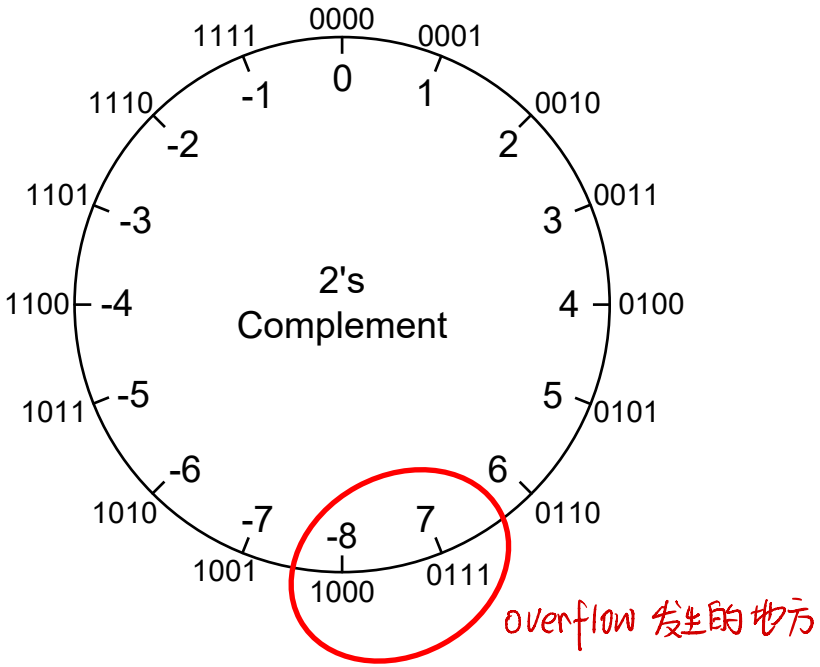
$$-X_{2C} = x_3' x_2' x_1' x_0' + 0001$$

$$\begin{array}{rcccc} & 1 & 0 & 0 & 0 \\ - & x_3 & x_2 & x_1 & x_0 \\ \hline -X_{2C} = & x_3' & x_2' & x_1' & x_0' \end{array} + 0001$$

$$-X_{2C} = 16 - X_{2C}$$

$$-X_{2C} = 2^n - X_{2C}$$

2's Complement Overflow

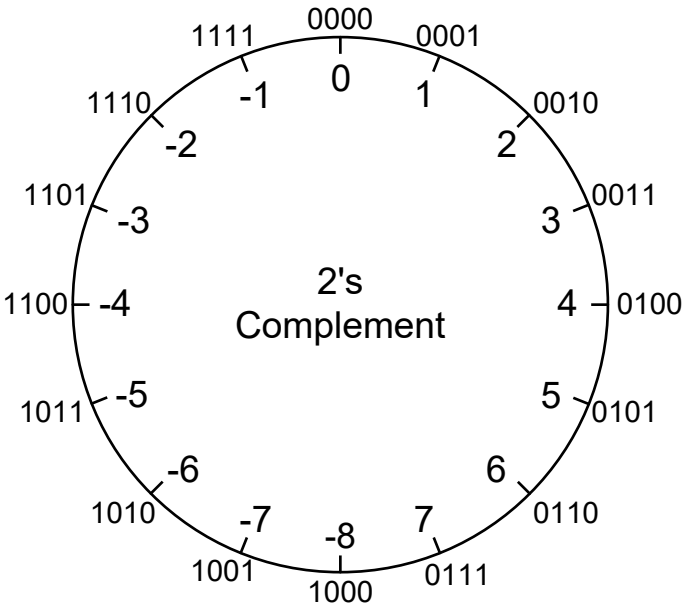


No Overflow: $A \geq 0, B < 0, A + B \in [-8, 7]$

Positive Overflew: $A, B \geq 0, A + B > 7$

Negative Overflow: $A, B < 0, A + B < -8$

2's Complement Overflow



$-8 : 1000$

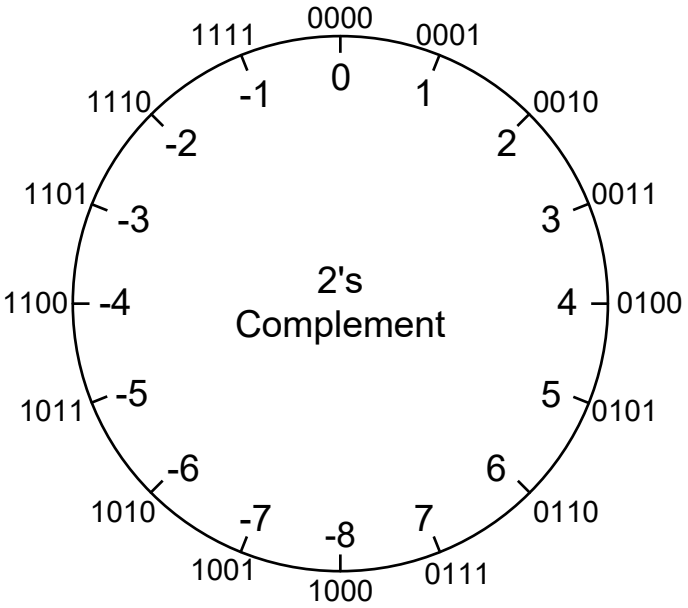
$6 : 0110$

$-8 + (-6) = -14$

overflow

	1	0	0	1	1
-8	1	0	0	0	
- +	1	0	0	1	
=	0	0	1	0	X incorrect answer

2's Complement Overflow



3 + 2 = 5

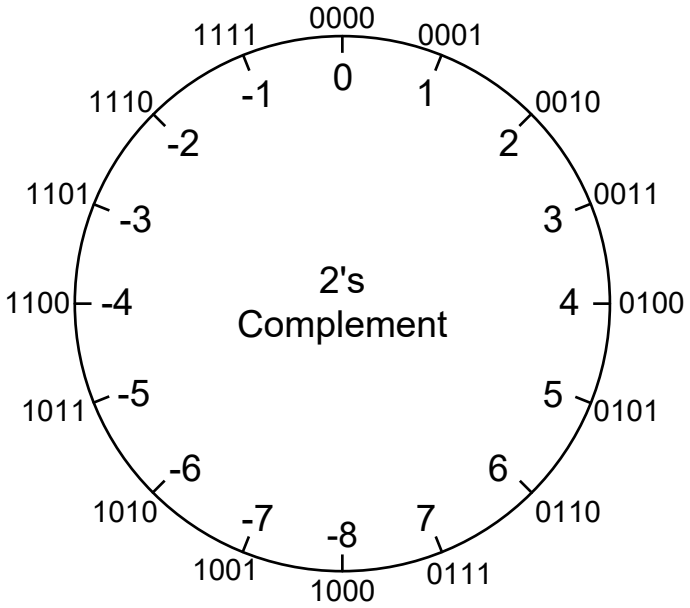
3: 0011

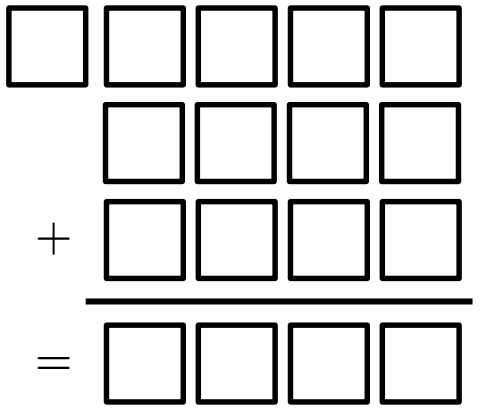
2: 0010

since we are adding

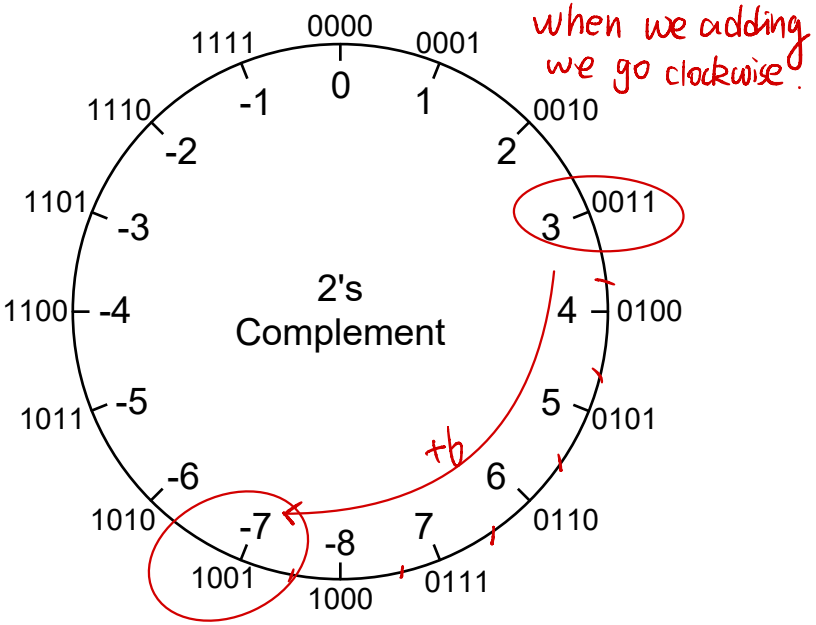
	0	0	1	0	0
		0	0	1	1
+		0	0	1	0
=		0	1	0	1

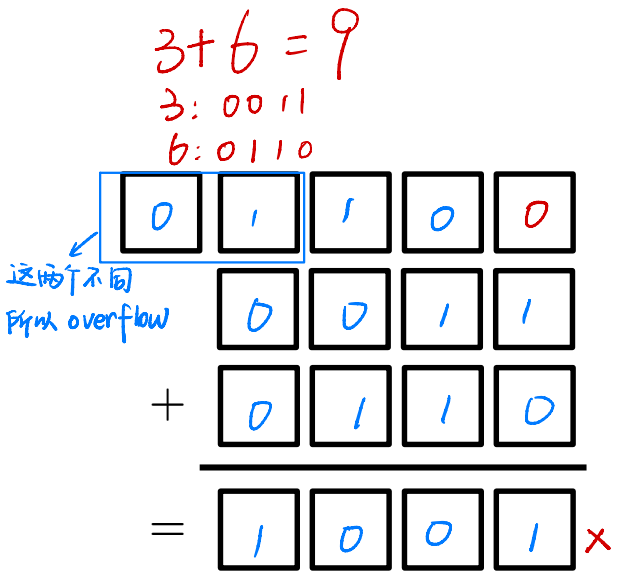
2’s Complement Overflow





2's Complement Overflow

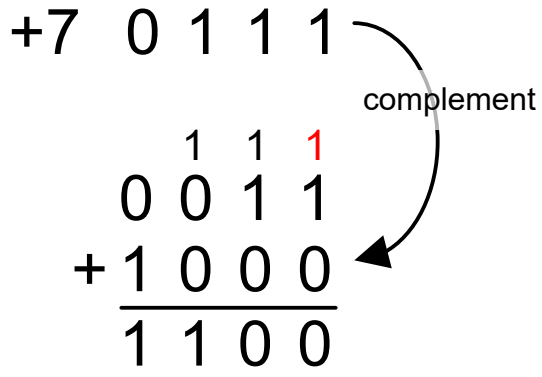




In Class Exercise

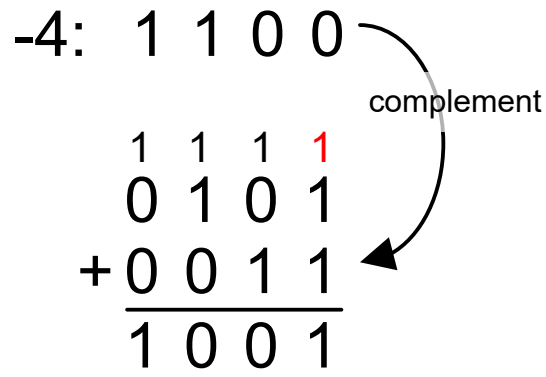
Perform the following operations in the same fashion as the ripple-carry adder/subtractor. Note whether or not overflow has occurred.

Subtract 7 from 3 (3 – 7)



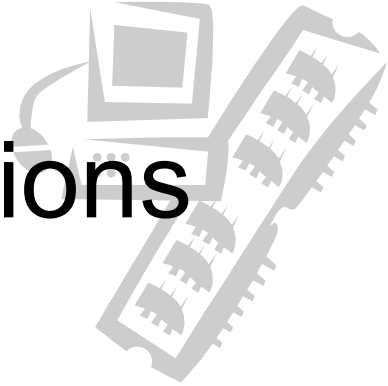
Carry-in to most-significant column is equal to carry-out of most-significant column → No overflow

Subtract -4 from 5 (5 – (-4))



Carry-in to most-significant column is not equal to carry-out of most-significant column → Overflow

Derivative Arithmetic Operations



- Incrementing (+1) and decrementing (-1)
  - Start with an adder and simplify the circuit
- Multiplying an unsigned by a power of two
  - Shift to the left
- Dividing an unsigned by a power of two
  - Shift to the right
- Multiplication: A*B
  - Same algorithm as in decimal
  - Partial products: multiply A by digits of B
  - Add up all partial products
- Division A/B: ?